Augmenting Types

# Aumentar tipos

JavaScript permite que los tipos básicos del lenguaje sean *augmented*. En el [Capítulo 3](ch03.html), vimos que añadir un método a Object.prototype hace que ese método esté disponible para todos los objetos. Esto también funciona para funciones, matrices, cadenas, números, expresiones regulares y booleanos.

Por ejemplo, aumentando Function.prototype, podemos hacer que un método esté disponible para todas las funciones:

Function.prototype.method = function (name, func) {  
 this.prototype[name] = func;  
 return this;  
};

Al aumentar Function.prototype con un método method, ya no tenemos que escribir el nombre de la propiedad prototype. Ahora podemos ocultar esa fealdad.

JavaScript no tiene un tipo entero independiente, por lo que a veces es necesario extraer sólo la parte entera de un número. El método que proporciona JavaScript para hacerlo es feo. Podemos arreglarlo añadiendo un método integera Number.prototype. Utiliza Math.ceil o Math.floor, dependiendo del signo del número:

Number.method('integer', function ( ) {  
 return Math[this < 0 ? 'ceil' : 'floor'](this);  
});  
  
document.writeln((-10 / 3).integer( )); // −3

JavaScript carece de un método que elimine los espacios de los extremos de una cadena. Es un descuido fácil de arreglar:

String.method('trim', function ( ) {  
 return this.replace(/^\s+|\s+$/g, '');  
});  
  
document.writeln('"' + " neat ".trim( ) + '"');

Nuestro método trim utiliza una expresión regular. Veremos mucho más sobre expresiones regulares en [el Capítulo 7](ch07.html).

Aumentando los tipos básicos, podemos mejorar significativamente la expresividad del lenguaje. Debido a la naturaleza dinámica de la herencia prototípica de JavaScript, todos los valores están dotados inmediatamente de los nuevos métodos, incluso los valores que se crearon antes de que se crearan los métodos.

Los prototipos de los tipos básicos son estructuras públicas, por lo que hay que tener cuidado al mezclar bibliotecas. Una técnica defensiva consiste en añadir un método sólo si se sabe que falta:

// Add a method conditionally.  
  
Function.prototype.method = function (name, func) {  
 if (!this.prototype[name]) {  
 this.prototype[name] = func;  
 return this;  
 }  
};

Otra preocupación es que la declaración for in interactúa mal con los prototipos. En el [Capítulo 3](ch03.html) vimos un par de formas de mitigarlo: podemos utilizar el método hasOwnProperty para filtrar las propiedades heredadas, y podemos buscar tipos específicos.